Lineárne vyhľadávanie

Vyhľadávanie je činnosť s cieľom zistiť, či sa prvok so zadanou vlastnosťou nachádza vo zvolenej skupine údajov.

Môžeme zisťovať:

- či sa prvok s požadovanou vlastnosťou v poli nachádza
- počet jeho výskytov
- miesta výskytov (prvé, posledné, všetky miesta)
- ak sa nenachádza, miesto, kde ho treba vložiť, aby si pole zachovalo pôvodné vlastnosti
- atď.

Dva základné spôsoby vyhľadávania:

- **lineárne** (sekvenčné), keď skúmame prvok za prvkom; používame ho pri vyhľadávaní v neutriedenom poli
- **binárne**, keď podľa hodnoty prvku v strede skúmanej oblasti pokračujeme v prehľadávaní buď dolnej alebo hornej časti oblasti; možno použiť len pri utriedenom poli.

Typické úlohy lineárneho vyhľadávania

Poznámky:

- V príkladoch predpokladáme, že pole má aspoň jeden prvok!
- V prvom bloku sa sústredíme na vyhľadávanie v poli obsahujúcom celé čísla (typ int).
- Odporúčame najprv zopakovať si v kapitole Príkazy minimálne časť Cykly.

Vyhľadávanie v poli celých čísel

Príklad L.1
Vytvorte funkciu, ktorá nájde a vráti najmenšiu hodnotu v poli. V Pythone funkcia min(zoznam).

```
def vratNajmensiPrvok(pole):
    min = pole[0]
    for i in range(1,len(pole)):
        if pole[i] < min: min = pole[i]
    return min
```

Použitie:

```
poleInt = vytvorPoleIntNahodne()
print("Najmenší prvok v poli:", vratNajmensiPrvok(poleInt))
print("Kontrola:", min(poleInt))
```

Príklad L.2
Vytvorte funkciu, ktorá nájde a vráti najväčšiu hodnotu v poli. V Pythone funkcia max(zoznam).
Vo funkcii `vratNajvacsiPrvok` sme, na rozdiel od funkcie `vratNajmensiPrvok`, nepoužili indexovú premennú vo for-cykle ale rezací operátor.

```
def vratNajvacsiPrvok(pole):
    max = pole[0]
    for prvok in pole[1:]:
        if prvok > max: max = prvok
    return max
```

Použitie:

```
poleInt = vytvorPoleIntNahodne()
print("Najväčší prvok v poli:", vratNajvacsiPrvok(poleInt))
print("Kontrola:", max(poleInt))
```

Prvý typ úloh lineárneho vyhľadávania je zistenie, či sa prvok s požadovanou vlastnosťou nachádza v poli.

Pri hľadaní najmenšej alebo najväčšej hodnoty v poli zrejme musíme prejsť celé pole (použiť for-cyklus) a pozor si treba dať na priradenie počiatočnej hodnoty minima a maxima, aby bola funkcia „univerzálna“ (napríklad pri priradení počiatočnej hodnoty max = 0 by funkcia pre polia len so zápornými hodnotami dávala zlý výsledok). Zrejme prvok s najväčšou alebo najmenšou hodnotou sa v poli určite nachádza, preto sme sa pýtali rovno na veľkosť hodnoty prvku s touto vlastnosťou. Ak sa pýtame na výskyt prvku s určitou vlastnosťou v poli vo všeobecnosti, máme na mysli prvý výskyt a po nájdení prvku s požadovanou vlastnosťou sa prehľadávanie poľa môže ukončiť, pričom funkcia vracia jednu z dvoch hodnôt - True alebo False (hodnota sa našla, vyskytla v poli, alebo nenašla). Python na riešenie takéhoto problému ponúka **operátor príslušnosti in**, ktorý sa používa v tvare „prvok in zoznam“, ktorý vráti jednu z hodnôt True alebo False. Takže triviálnym riešením problému, či sa prvok s požadovanou vlastnosťou nachádza v poli, je v Pythone napríklad skupina príkazov

```
poleInt = vytvorPoleIntNahodne()
hodnota = int(input("V poli hľadať hodnotu: "))
if hodnota in poleInt:
    print(hodnota,"sa v poli vyskytuje!")
else:
    print(hodnota,"sa v poli nevyskytuje!")
```

Pre zlepšenie algoritmického myslenia ďalej predpokladajme, že nemáme k dispozícii operátor príslušnosti in!

Príklad L.3
Vytvorte funkciu, ktorá zistí, či sa v poli nachádza nulová hodnota. Použijeme riešenie s indexmi, pri ktorom si teba dať pozor, aby sme s indexom „nevyleteli“ z poľa, t.j. neskúmali prvok `pole[len(pole)]`[1], ktorý už v poli neexistuje! Preto odporúčame používať riešenie, pokiaľ to je možné, uvedené v príklade L.4.

```
def vratVyskytNuly(pole):
    i = 0
    while pole[i] != 0 and i < len(pole)-1:
        i += 1
    return pole[i] == 0
```

NEODPORÚČAME POUŽÍVAŤ!

Použitie:

```
poleInt = vytvorPoleIntNahodne()
if vratVyskytNuly(poleInt):
    print("Nula sa v poli vyskytuje!")
else:
    print("Nula sa v poli nevyskytuje!")
```

Príklad L.3 zovšeobecnime na zistenie, či sa v poli nachádza hľadaná hodnota. Použijeme v súčasnosti častejšie používaný zápis (bez „špekulácie“ len(pole)-1) pri ktorom nehrozí „vybehnutie z poľa“.

Príklad L.4
Vytvorte funkciu, ktorá zistí, či sa v poli nachádza zadaná hľadaná hodnota. V Pythone operátor príslušnosti in.

„Klasické“ riešenie (bez operátora in) while-cyklom a s indexovanou premennou pole[i]:

```
def vratVyskytHodnotyCW(pole, hladat):
    i = 0
    while i < len(pole):
        if pole[i] == hladat:
            return True
        i += 1
    return False
```

Použitie

```
poleInt = vytvorPoleIntNahodne()
```

[1] Niektoré programovacie jazyky používajú funkciu High(pole), ktorá vracia hodnotu najvyššieho dovoleného indexu pre pole , t.j. High(pole) = Length(pole)-1.

```
hladat = int(input("Hľadať hodnotu: "))
if vratVyskytHodnotyCW(poleInt, hladat):
    print("Hodnota", hladat,"sa v poli vyskytuje!")
else:
    print("Hodnota", hladat,"sa v poli nevyskytuje!")
print("Kontrola:", hladat in poleInt)
```

Ďalšie úlohy na precvičenie:
Vytvorte a použite funkciu, ktorá efektívne

- zistí, či sa v celočíselnom poli nachádza párne (nepárne) číslo
- zistí, či sa v celočíselnom poli nachádza číslo deliteľné zadaným číslom
- zistí, či sa v celočíselnom poli nachádza záporná hodnota
- zistí, či sa v celočíselnom poli nachádza číslo zo zadaného intervalu

Ďalší typ úloh lineárneho vyhľadávania znamená zistiť, koľkokrát sa hľadaná hodnota nachádza v poli, pričom sa v poli nemusí vôbec vyskytnúť. V Pythone to je funkcia zoznam.count(hodnota).

Príklad L.5
Vytvorte funkcia, ktorá vráti počet výskytov hľadanej hodnoty v poli. Nepoužite funkciu pole.count(hladat).

Úloha si vyžaduje prejsť celé pole, takže „je šitá“ na for-cyklus, či už s použitím indexovanej premennej pole[i] a generátora range alebo for-cyklom „prvok in pole“.

```
def vratPocetVyskytov(pole, hladat):
    pocet = 0
    for prvok in pole:
        if prvok == hladat:
            pocet += 1
    return pocet
```

Použitie:

```
poleInt = vytvorPoleIntNahodne()
hladat = int(input("Hľadať hodnotu: "))
print("Počet výskytov hodnoty {} v poli: {}".format(hladat, vratPocetVyskytov(poleInt, hladat)))
print("Kontrola:", poleInt.count(hladat))
```

Ďalšie úlohy na precvičenie:
Vytvorte a použite funkciu, ktorá efektívne

- zistí počet párnych (nepárnych) čísel v celočíselnom poli
- zistí počet čísel deliteľných zadaným číslom v celočíselnom poli
- zistí počet záporných hodnôt v celočíselnom poli
- zistí počet čísel zo zadaného intervalu v celočíselnom poli

Ďalší typ úloh lineárneho vyhľadávania znamená zistiť miesto - presnejšie index, výskytu hľadanej hodnoty. Zisťovať sa môže index prvého výskytu zľava hľadanej hodnoty v poli, index posledného výskytu a indexy všetkých výskytov hľadanej hodnoty.

Príklad L.6
Vytvorte funkciu, ktorá vráti index prvého výskytu zľava hľadanej hodnoty v poli. V Pythone funkcia zoznam.index(hodnota).

„Klasické“ riešenie while-cyklom:

```
def vratIndexPrvehoVyskytuCW(pole, hladat):
    i = 0
    while i < len(pole):
        if pole[i] == hladat:
            return i
```

```
        i += 1
    return -1
```

Použitie:

```
poleInt = vytvorPoleIntNahodne()
hladat = int(input("Hľadať hodnotu: "))
vysledok = vratIndexPrvehoVyskytuCW(poleInt, hladat)
if vysledok != -1:
    print("Prvý výskyt hodnoty", hladat,"je na indexe",vysledok)
else:
    print(hladat,"sa v poli nevyskytuje!")
print("Kontrola: ", end="")
if hladat in poleInt:
    print("Prvý výskyt hodnoty", hladat,"je na indexe",poleInt.index(hladat))
else:
    print(hladat,"sa v poli nevyskytuje!")
```

Riešenie for-cyklom:

```
def vratIndexPrvehoVyskytu(pole, hladat):
    """ vráti index prvého výskytu hľadanej hodnoty alebo -1 """
    for i in range(len(pole)):
        if pole[i] == hladat:
            return i
    return -1
```

Použitie:

```
poleInt = vytvorPoleIntNahodne()
hladat = int(input("Hľadať hodnotu: "))
index = vratIndexPrvehoVyskytu(poleInt, hladat)
if index != -1:
    print("Prvý výskyt hodnoty", hladat,"je na indexe",index)
else:
    print(hladat,"sa v poli nevyskytuje!")
print("Kontrola: ", end="")
if hladat in poleInt:
    print(poleInt.index(hladat))
else:
    print(hladat,"sa v poli nevyskytuje!")
```

POZOR! Ak by sme sa inšpirovali pri riešení ostatného problému algoritmom príkladu L.4, nemusíme uspieť:

```
def vratIndexPrvehoVyskytuZLE1(pole, hladat):
    i = 0
    while i < len(pole):
        if pole[i] == hladat:
            return i
        i += 1
    return False
```

N E P O U Ž Í V A Ť !

rovnako zle:

```
def vratIndexPrvehoVyskytuZLE2(pole, hladat):
    for i in range(len(pole)):
        if pole[i] == hladat:
            return i
    return False
```

N E P O U Ž Í V A Ť !

Použitie:

```
poleInt = vytvorPoleIntNahodne()
hladat = int(input("Hľadať hodnotu: "))
vysledok = vratIndexPrvehoVyskytuZLE1(poleInt, hladat)
if vysledok:
    print("Prvý výskyt hodnoty", hladat,"je na indexe",vysledok)
else:
```

```
    print(hladat,"sa v poli nevyskytuje!")
print("Kontrola: ", end="")
if hladat in poleInt:
    print("Prvý výskyt hodnoty", hladat,"je na indexe",poleInt.index(hladat))
else:
    print(hladat,"sa v poli nevyskytuje!")
```

Nepomôže ani napríklad výstup:

```
if vysledok == False:
    print(hladat,"sa v poli nevyskytuje!")
else:
    print("Prvý výskyt hodnoty", hladat,"je na indexe",vysledok)
```

Trochu vám pri hľadaní chyby pomôžeme výpisom.

Výpis:

```
Počet prvkov: 10
Prvky poľa:
[5, 7, 0, 4, 2, 6, 7, 2, 2, 6]
Hľadať hodnotu: 5
5 sa v poli nevyskytuje!
Kontrola: Prvý výskyt hodnoty 5 je na indexe 0
```

Príklad L.7

Vytvorte funkciu, ktorá vráti index posledného výskytu hľadanej hodnoty v poli.

„Klasické“ riešenie while-cyklom:

```
def vratIndexPoslednehoVyskytuCW(pole, hladat):
    """ vráti index posledného výskytu hľadanej hodnoty alebo -1 """
    i = len(pole)
    while i > 0:
        i -= 1
        if pole[i] == hladat:
            return i
    return -1
```

Použitie:

```
poleInt = vytvorPoleIntNahodne()
hladat = int(input("Hľadať hodnotu: "))
index = vratIndexPoslednehoVyskytuCW(poleInt, hladat)
if index != -1:
    print("Posledný výskyt hodnoty", hladat,"je na indexe",index)

else:
    print(hladat,"sa v poli nevyskytuje!")
print("Kontrola - počet výskytov:", poleInt.count(hladat))
```

Riešenie for-cyklom. Prečo pracujeme s indexom i-1?

```
def vratIndexPoslednehoVyskytu(pole, hladat):
    """ vráti index posledného výskytu hľadanej hodnoty alebo -1 """
    for i in range(len(pole),0,-1):
        if pole[i-1] == hladat:
            return i-1
    return -1
```

Použitie:

```
poleInt = vytvorPoleIntNahodne()
hladat = int(input("Hľadať hodnotu: "))
index = vratIndexPoslednehoVyskytu(poleInt, hladat)
if index != -1:
    print("Posledný výskyt hodnoty", hladat,"je na indexe",index)
else:
    print(hladat,"sa v poli nevyskytuje!")
```

Ako sme uvažovali pri písaní nasledujúcej funkcie, ktorá tiež vráti index posledného výskytu hľadanej hodnoty v poli alebo číslo -1?

```
def vratIndexPoslednehoVyskytuNeef(pole, hladat):
    """ vráti index posledného výskytu hľadanej hodnoty alebo -1 """
    index = -1
    for i in range(len(pole)):
        if pole[i] == hladat:
            index = i
    return index
```

Príklad L.8
Vytvorte funkciu, ktorá vráti indexy všetkých výskytov hľadanej hodnoty v poli.

„Klasické“ riešenie while-cyklom:

```
def vratIndexyVsetkychVyskytovCW(pole, hladat):
    indexy = []
    i = 0
    while i < len(pole):
        if pole[i] == hladat:
            indexy.append(i)
        i += 1
    return indexy
```

Použitie:

```
poleInt = vytvorPoleIntNahodne()
hladat = int(input("Hľadať hodnotu: "))
print("Všetky indexy:", vratIndexyVsetkychVyskytovCW(poleInt, hladat))
print("Kontrola:", [index for index, hodnota in enumerate(poleInt) if hodnota == hladat])
```

Riešenie for-cyklom:

```
def vratIndexyVsetkychVyskytov(pole, hladat):
    """ vráti indexy všetkých výskytov hľadanej hodnoty alebo prázdny [] """
    indexy = []
    for i in range(len(pole)):
        if pole[i] == hladat:
            indexy.append(i)
    return indexy
```

Použitie:

```
poleInt = vytvorPoleIntNahodne()
hladat = int(input("Hľadať hodnotu: "))
print("Všetky indexy:", vratIndexyVsetkychVyskytov(poleInt, hladat))
print("Kontrola:", [index for index, hodnota in enumerate(poleInt) if hodnota == hladat])
```

Všimnite si použitie funkcie enumerate[1] v kontrolnom výpise v oboch ostatných použitiach.
Výpis:

```
Počet prvkov: 15
Prvky poľa:
[7, 3, 4, 2, 4, 3, 4, 3, 0, 4, 8, 2, 4, 0, 5]
Hľadať hodnotu: 4
Všetky indexy: [2, 4, 6, 9, 12]
Kontrola: [2, 4, 6, 9, 12]
```

[1] Funkcia enumerate, zjednodušene povedané, vytvorí dvojice index – hodnota zo skupiny hodnôt. Pri použití funkcie enumerate v cykle for pracujeme s dvoma premennými, jedna obsahuje index položky a druhá samotnú položku. Celý vyššie uvedený zápis možno interpretovať ako vytvorenie zoznamu, ktorý obsahuje indexy získané funkciou enumerate, pre ktoré sa hodnota rovná hľadanej.

Príklad L.9*
Vytvorte funkciu, ktorá vráti indexy všetkých výskytov hľadanej hodnoty v poli s využitím funkcie z príkladu L.6: `vratIndexPrvehoVyskytu(pole, hladat)` alebo štandardnej funkcie zoznam.index(hodnota).

Autorkou riešenia je študentka Natália Holková.

```
def vratIndexyVsetkychVyskytov(pole, hladat):
    poleVyskytov = []
    i = 0
    while True:
        index = vratIndexPrvehoVyskytu(pole, hladat)
        if index != -1:
            poleVyskytov.append(index+i)
            pole = pole[index+1: ]
            i += index+1
        else:
            break
    return poleVyskytov
```

Využitie štandardnej funkcie zoznam.index(hodnota) (ak sa hodnota v zozname nevyskytuje, funkcia vyvolá výnimku ValueError):

```
def vratIndexyVsetkychVyskytovFuindex(pole, hladat):
    poleVyskytov = []
    i = 0
    while True:
        try:
            inx = pole.index(hladat)
        except ValueError:
            return poleVyskytov
        poleVyskytov.append(inx+i)
        pole = pole[inx+1: ]
        i += inx+1
```

Použitie:
```
poleInt = vytvorPoleIntNahodne()
hladat = int(input("Hľadať v poli hodnotu: "))
print("Všetky indexy výskytov: {}".format(vratIndexyVsetkychVyskytovFuindex(poleInt, hladat)))
print("Kontrola:", [index for index, hodnota in enumerate(poleInt) if hodnota == hladat])
```
Výpis:
```
Počet prvkov: 15
Prvky poľa:
[1, 5, 6, 1, 5, 6, 3, 5, 9, 0, 0, 3, 5, 0, 2]
Hľadať v poli hodnotu: 5
Všetky indexy výskytov: [1, 4, 7, 12]
Kontrola: [1, 4, 7, 12]
```

Príklad L.10
Vytvorte funkciu, ktorá zistí, či sa zadaná hodnota nachádza v poli a ak nie, vloží ju na koniec poľa.

Na vykonanie funkcie sa teraz možno pozerať odlišne ako v predchádzajúcich príkladoch. Všetky predchádzajúce funkcie vracali nejakú novú informáciu (False/True, index alebo indexy), ktorú sme ďalej spracúvali a preto uvádzali volanie funkcie nie samostatne (ako jeden príkaz v riadku) ale v príkazoch, kde sme chceli získanú hodnotu ďalej použiť.

V príklade L.10, ak sa hľadaná hodnota v poli nachádza, nemá sa pole zmeniť, t.j. nemá sa nič vykonať; ak sa hľadaná hodnota nenachádza v poli, treba ju „len“ pridať na koniec existujúceho poľa - vykonať jeden príkaz.
Z viacerých alternatív sme zvolili „triviálne pythonovské“ riešenie a „klasické“ riešenie while-cyklom.

```
def vlozAkSaNevyskytla(pole, hladat):
    if hladat not in pole:
        pole.append(hladat)
```

Použitie:
```
poleInt = vytvorPoleIntNahodne()
hladat = int(input("Hľadať hodnotu: "))
vlozAkSaNevyskytla(poleInt, hladat)
print(poleInt)
```
Výpis:
```
Počet prvkov: 15
Prvky poľa:
[7, 0, 0, 5, 9, 8, 1, 0, 3, 6, 3, 9, 4, 5, 3]
Hľadať hodnotu: 2
[7, 0, 0, 5, 9, 8, 1, 0, 3, 6, 3, 9, 4, 5, 3, 2]
```

„Klasické“ riešenie while-cyklom:

```
def vlozAkSaNevyskytlaCW(pole, hladat):
    i = 0
    while i < len(pole):
        if pole[i] == hladat:
            break
        i += 1
    if i == len(pole):
        pole.append(hladat)
```

Použitie:
```
poleInt = vytvorPoleIntNahodne()
hladat = int(input("Hľadať hodnotu: "))
vlozAkSaNevyskytlaCW(poleInt, hladat)
print(poleInt)
```
Prvý výpis:
```
Počet prvkov: 10
Prvky poľa:
[2, 0, 6, 2, 9, 3, 8, 5, 5, 9]
Hľadať hodnotu: 9
[2, 0, 6, 2, 9, 3, 8, 5, 5, 9]
```
Druhý výpis:
```
Počet prvkov: 10
Prvky poľa:
[6, 1, 6, 9, 9, 4, 4, 0, 2, 7]
Hľadať hodnotu: 5
[6, 1, 6, 9, 9, 4, 4, 0, 2, 7, 5]
```

Dôležité!
Volanie funkcie **`vlozAkSaNevyskytla(poleInt, hladat)`** je v oboch riešeniach pre nás neštandardné. Doteraz vždy, keď sme chceli pomocou funkcie niečo zmeniť, sme novú hodnotu vyviezli cez príkaz return. Volanie funkcie sme uvádzali na miestach, kde sme chceli dosadiť výsledok funkcie, t.j. buď na pravej strane príkazu priradenia alebo napríklad v príkaze výstupu print. V týchto situáciách hovoríme, že je volanie funkcie v úlohe **funkčného výrazu** (hodnota získaná vykonaním príkazov funkcie sa na mieste funkčného výrazu - volania funkcie, dosadí do príkazu).
V ostatných dvoch riešeniach je však volanie funkcie **`vlozAkSaNevyskytla(poleInt, hladat)`** na úrovni príkazu, pretože je samostatne na riadku (!) a zmena získaná vykonaním príkazov funkcie sa nikde viditeľne neprirađuje - neobsahuje návratový príkaz return!
Vysvetlenie je nasledujúce. Ak sú parametrami funkcie meniteľné typy (v našom prípade pole resp. list), pri niektorých operáciách zmeny vykonávané s formálnymi parametrami vo funkcii sa automaticky vykonávajú aj so skutočnými parametrami - globálnymi premennými, s ktorými bola funkcia volaná. Konkrétne v našom prípade, ak do lokálnej premennej pole pridáme na koniec poľa hľadanú hodnotu, táto bude automaticky pridaná aj na koniec polaInt, teda zmena sa prejaví mimo funkcie aj bez zrejmého vyvezenia hodnoty formálneho parametra pole, resp. použitia príkazu return! Pri týchto meniteľných údajových typoch ako parametroch hovoríme, že nešlo k odovzdaniu konkrétnej hodnoty skutočného parametra formálnemu (ako pre nemeniteľných typoch), ale že formálnemu parametru bol odovzdaný odkaz (referencia) na skutočný parameter. V skutočnosti to znamená, že tieto dve premenné (formálny a skutočný parameter) reprezentujú - „ukazujú na“ tú istú skupinu dát. V praxi to

znamená, že robiť niektoré zmeny v meniteľnom (mutable) type dát možno pomocou ktorejkoľvek premennej „ukazujúcej“ na takýto typ dát.
Zmeny, ktoré sa prejavia „v oboch“[1] poliach - prenesú sa referenciou od dvoch premenných, sú napríklad zmena hodnoty prvku poľa, odstránenie prvku alebo rezu z poľa (pomocou príkazu del) a iné.
Záver:
Je dobré kontrolovať, či použitá funkcia s referencovanými parametrami[2] použila príkazy, ktoré zmenili alebo nezmenili aj globálne pole (skutočný parameter funkcie). Kontrolu možno jednoducho vykonať napríklad výpisom globálneho poľa pred a po volaní funkcie.
Pokiaľ nechceme, aby sa zmeny prejavili v zdrojovom poli, musíme vytvoriť kópiu zdrojového poľa a tú použiť ako skutočný parameter funkcie.
Poznámka:
Volanie funkcie na úrovni príkazu sme použili hneď v úvode študijného textu Funkcie a moduly, funkcie sme však volali bez parametrov (funkcia vypisNadpis()) alebo „s jednoduchým“ (imutable) parametrom typu str.

Napríklad:

```
import pole                                  # použitie modulu pole, pozri študijný text Funkcie a moduly

def vlozAkSaNevyskytla(pole, hladat):
    if hladat not in pole:
        pole.append(hladat)

poleInt = pole.vytvorIntNahodne(1,9)
poleIntZmenene = poleInt[:]                  # vytvorenie kópie zdrojového poľa
vlozAkSaNevyskytla(poleIntZmenene, 10)
print("Pôvodné pole:", poleInt)
print("Zmenené pole:", poleIntZmenene)
```

Vypíše:

```
Počet prvkov poľa: 10
Prvky poľa:
[1, 7, 9, 9, 7, 7, 3, 4, 9, 8]
Pôvodné pole: [1, 7, 9, 9, 7, 7, 3, 4, 9, 8]
Zmenené pole: [1, 7, 9, 9, 7, 7, 3, 4, 9, 8, 10]
```

Úlohy na precvičenie:

- vytvorte funkciu, ktorá zistí, či sa zadaná hodnota nachádza v poli a ak nie, vloží ju na začiatok poľa (spravte „triviálne“ a netriviálne riešenie)
- v predchádzajúcich príkladoch riešte úlohy bez použitia funkcie zoznam.append(hodnota), t.j. nahraďte ju elementárnejšou operáciou spájania +.

Vyhľadávanie v poli reálnych čísel

Vyhľadávacie algoritmy z ostatného celku Vyhľadávanie v poli celých čísel môžeme aplikovať aj na pole reálnych čísel. Podstatný rozdiel pri práci s celými a reálnymi číslami je v tom, že pokiaľ „sa pohybujeme“ (počítame) len v intervale celých čísel zobraziteľných v počítači, všetky výpočty sú absolútne presné. Pri práci s reálnymi číslami toto tvrdenie neplatí. Kvôli konečnej pamäti počítača principiálne nemôže zaznamenať v počítači všetky reálne čísla (jednak ich je nekonečne veľa, aj medzi ľubovoľnými dvoma reálnymi číslami, a pri reálnych číslach s neukončeným desatinným rozvojom, t.j. nekonečným počtom cifier za desatinnou čiarkou, ich nevieme ani presne uložiť do pamäte počítača).
Upozorníme len na problém zisťovania rovnosti dvoch reálnych čísel. Po analýze nasledujúceho programu zistíme, že vypíše číslo, pre ktoré už počítač nie je schopný zaznamenať prírastok eps > 0, pretože si menšie číslo nevie uložiť do pamäte; počítač všetky kladné čísla z intervalu <eps,0) vyhodnotí ako nulu. Ak by sme teda porovnávali dve rôzne reálne čísla, medzi ktorými je rozdiel v absolútnej hodnote menší alebo rovný eps, nezistíme rozdiel - pre počítač sú tieto dve čísla rovnaké.

[1] Pole je v pamäti len jedno, môže však na neho odkazovať viacej premenných.
[2] Referencované parametre sú parametre, ktorých typy sú mutable, t.j. meniteľné.

```
eps = 1
while True:
    eps = eps/2
    if 1.0 + eps == 1.0:
        break
print("Strojová nula:",eps)
```
Výpis:
```
Strojová nula: 1.1102230246251565e-16
```
symbol „e“ znamená exponent a číta sa „10 na“, t.j. je to číslo 0,0000000000000001110223024625156 5.
Môžete sa „pohrať“:
```
>>> 1 + 1e-16 == 1
True
>>> 1 + 1e-15 == 1
False
```
Musíme tiež povedať, že nemáme zaručené, či vidíme, t.j. či sú zobrazené, všetky cifry čísla uloženého v pamäti počítača.
Preto výraz `prvok == hladat` zapisujeme `abs(prvok - hladat) <= presnost`. Aplikované na príklad L.4:

```
import pole

def vratVyskytHodnotyCW(pole, hladat, presnost):
    i = 0
    while i < len(pole):
        if abs(pole[i] - hladat) <= presnost:
            return True
        i += 1
    return False

poleReal = pole.vytvorRealNahodne()
print(vratVyskytHodnotyCW(poleReal,0.5,0.1))
```

Poznámka:
Použitie operátora príslušnosti in v poli reálnych čísel môže byť problematické.

Vyhľadávanie v poli reťazcov

Vyhľadávacie algoritmy z celku Vyhľadávanie v poli celých čísel môžeme aplikovať aj na pole obsahujúce ako prvky znaky alebo reťazce. Porovnávanie znakov sa realizuje podľa poradia znakov v kódovacej tabuľke Unicode.
Znakové reťazce môžeme porovnávať pomocou relačných operácií (==, !=, <, >, <=, >=, is, not is). Reťazce sú usporiadané pomocou tzv. lexikografického usporiadania. Napr. platí
'abc' > 'ABC'
'Adam' < 'Eva'
'jana' < 'jano'
'Jana' < 'jana'
'Jana' < 'Ján'
'0' < 'A' < 'a' < 'Á'
Postupne sa pri tom porovnáva znak za znakom: kým sú v oboch reťazcoch rovnaké, pokračuje sa v porovnávaní; keď sa narazí na rozdielne znaky, tak sa tieto dva porovnajú navzájom a podľa toho sa nastaví celkový výsledok porovnania. Porovnávanie dvoch znakov sa robí podľa poradia znakov v kódovacej tabuľke Unicode.
Ako príklad uvediem použitie funkcie vratIndexyVsetkychVyskytov(), v ktorej nebolo treba nič zmeniť.

```
#import pole
def vratIndexyVsetkychVyskytov(pole, hladat):
    """ vráti indexy všetkých výskytov hľadanej hodnoty alebo prázdny [] """
    indexy = []
    for i in range(len(pole)):
        if pole[i] == hladat:
            indexy.append(i)
    return indexy
```

Použitie:

```
poleZnakov = pole.vytvorCharNahodne()
hladat = input("Hľadať hodnotu: ")
print("Všetky indexy:", vratIndexyVsetkychVyskytov(poleZnakov, hladat))
print("Kontrola:", [index for index, hodnota in enumerate(poleZnakov) if hodnota == hladat])
```

Výpis:

```
Počet prvkov poľa: 30
Prvky poľa:
['R', 'D', 'N', 'V', 'L', 'G', 'J', 'F', 'H', 'N', 'Y', 'P', 'S', 'U', 'K', 'G', 'C',
'G', 'L', 'U', 'E', 'J', 'Q', 'C', 'V', 'G', 'E', 'N', 'A', 'U']
Hľadať hodnotu: G
Všetky indexy: [5, 15, 17, 25]
Kontrola: [5, 15, 17, 25]
```

Úlohy:

- všetky najpoužívanejšie vyhľadávacie algoritmy si odskúšajte na poliach reálnych čísel, znakov aj reťazcov.